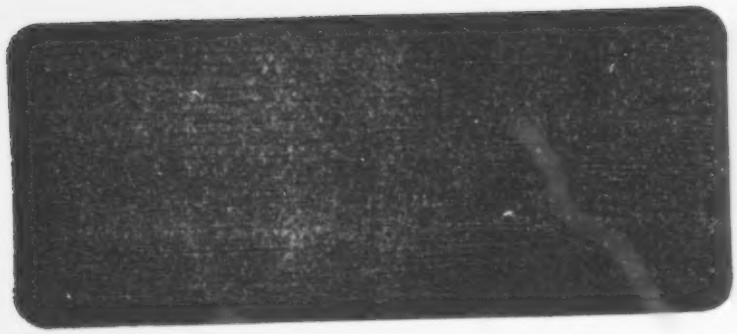
98-623



company

y

intelligent

technology

Applied Silicon Inc. Canada

220 - 2427 Holly Lane. Ottawa. Ontario K1V 7P2 • Tel: (613) 738-2434 • Fax: (613) 738-075(

CONTRACT NUMBER W7714-5-9868

PORTABLE INTERFERENCE FREQUENCY SOURCES

REMOTE CONTROL UPGRADE FINAL REPORT

Contract Number W7714-5-9868

Revision 1.0

prepared for: Department of National Defence
Defence Research Establishment Ottawa
3701 Carling Avenue
Ottawa, Ontario
K1A 0K2

Scientific Authority: Denis Lamothe
(613) 998-2040

by
John Curnow

APPLIED SILICON INC. CANADA
220-2427 HOLLY LANE
OTTAWA, ONTARIO, CANADA
K1V 7P2
(613) 738-2434

21 August 1995

Document #974-55126-02

TABLE OF CONTENTS

1.0 INTRODUCTION 1

2.0 HAND-HELD TERMINAL STORAGE 1

3.0 ENCLOSURE UNDERSIDE LAYOUT 2

4.0 ANTENNA AND MODEM MOUNTS 3

5.0 COMPUTER BOARDS 3

6.0 INTERNAL COMPONENT LAYOUT 3

7.0 SOFTWARE 5

8.0 POWER SUPPLY BOARD CHANGES 5

9.0 TESTING 5

9.1 Remote Control Tests 6
9.2 Testing the Second PIFS 7
9.3 Cooling Test 7
9.4 Battery Load Test 7

10.0 PHOTOGRAPH DESCRIPTIONS 8

11.0 CONCLUSIONS 10

PHOTOGRAPHS

APPENDIX A - CONTROL PROGRAM LISTING

APPENDIX B - CIRCUIT SCHEMATICS

LIST OF FIGURES

FIGURE NUMBER	*TITLE*	*PAGE*
1	PIFS Internal Component Layout	8

1.0 INTRODUCTION

This document is the Final Report for the Portable Interference Frequency Sources (PIFS) remote control modifications work. This task is covered under section 2.1 of the Statement of Work for the W7714-5-9868 contract.

A Design Specification Document (DSD) was written in early July 1995. It covered the hardware and software design of the PIFS modifications in detail. A Preliminary Design Specification Document was written before the main DSD. It briefly covered the modifications design of the PIFS and allowed the ordering of parts. The DSD was written after most of the design had been executed, for both the hardware and software. Therefore, it is an accurate design document, with minor changes and exceptions. These changes and exceptions are detailed in this final report. Most of the changes are mechanical. Since so few changes have been made since the DSD was written, the information contained in it will not be repeated in this document. For a complete final detailed design the reader should read the DSD before reading this final report.

The PIFS are two identical units which act as far field RF beacon sources for the Experimental Array Radar System (EARS). They are also known as "jammers" since their primary task is to jam the transmitted X-band radar returns of the EARS. They were originally designed and built to be manually controlled, requiring constant supervision by an operator. The operator would change the PIFS parameters using manual switches, under direction from the main EARS operator via two-way radios. One of the manually controlled PIFS was used in radar trials at DREO Area Five with good results.

The remote control modifications allow direct control of the PIFS by the system controller computer. The PIFS operation is therefore synchronized with the EARS operation at all times. Personnel are required to set the PIFS on location in the radar test range and to turn the PIFS on. The PIFS is then controlled by the system controller, without intervention by a local PIFS operator. When the day's radar trials are complete, the PIFS is turned off and retrieved. As of the writing of this document, the control from the system controller has not been implemented. This control is beyond the scope of this task. A desktop PC has been used to remotely control the PIFS from the EARS lab in the same way that the system controller will.

2.0 HAND-HELD TERMINAL STORAGE

The model 80T45 hand held terminal is operated exactly as described in the DSD. It was undecided at the time of the writing of the DSD how the terminal would be physically stored and used. An external box has been mounted on the outside of the PIFS enclosure to hold the terminal.

The box is made of sheet aluminum and is 25.7 cm wide by 11.8 cm high by 3.7 cm deep. It is mounted on the side of the PIFS, beside the switch panel, on three 1.3 cm spacers. The front cover is hinged at the bottom edge. Two quarter-turn fasteners are used in the top corners to secure the front. The cover is gasketed to prevent ingress of precipitation. Rubber sheets are glued to the inside of the box to cushion the terminal. The box is large enough to easily contain the terminal and its coiled cord during storage and transportation. The terminal is removed for use and the cord is plugged into the modular jack on the switch connector panel. A bent sheet metal holder has been welded onto the front of the terminal storage box. The terminal can be put into this holder and used, with access to the keys. The holder is angled so the terminal is held at about a 20 degree angle from vertical. This position is suitable when the PIFS is used in an accessible position, up to a height of about two meters. The terminal storage box and holder have been painted orange, the same colour as the main PIFS enclosure. Photographs three and four show the terminal box and holder.

3.0 ENCLOSURE UNDERSIDE LAYOUT

As mentioned in the DSD, there are heat sinks mounted on the underside of the enclosure. The main heat sink is 34.2 cm long by 18.6 cm wide by 3.4 cm deep. It has straight fins which run the length of the heat sink and are parallel to the longest side of the enclosure. All the hot components are mounted directly above this heat sink. Heat sink compound is used between the heat sink and the underside plate and between each component and the plate.

Two other smaller heat sinks are mounted on the underside beside the main heat sink. They are identical black anodize commercial heat sinks. Each one has an LM150K voltage regulator mounted on it. The voltage regulators are mounted upside-down, with their pins protruding into the PIFS enclosure through 4.6 cm diameter holes. These heat sinks are each 10.5 cm by 7.6 cm by 2.5 cm deep. The three heat sinks are made of aluminum. Photograph six shows the heat sinks.

The DSD mentions the use of two rails mounted on the underside to protect the heat sinks. This has been changed. There are two 'legs', each made of 5/8 inch solid aluminum rod. The rods have two bends to form a U-shaped piece which is 47.5 cm long by 8.3 cm high. The ends of the rods are screwed to the bottom near each corner. The rods are lighter than the proposed rails and allow greater air movement under the enclosure. Care must be taken to sit the PIFS on a surface so that it is supported on the rods. This happens with a flat surfa but an uneven surface such as rocks could present a problem. otographs one, three, four and six show the support rods.

4.0 ANTENNA AND MODEM MOUNTS

The radio modem is mounted on the underside of the lid. It is secured using a piece of aluminum sheet metal which has been bent into a 'U' shape. The metal bracket is 14 cm long and runs most of the length of the modem. Strips of rubber are used between the bracket and modem to hold and cushion it. The modem is mounted on the lid beside the antenna mounting flange and above the CTI synthesizer. There is an air gap of seven centimeters between the top of the synthesizer and the modem bracket. This is adequate for air flow and cooling. Photograph seven shows the mounting of the modem.

The antenna mount is similar to the flange described in the DSD except it is only 3.3 cm tall. It has a screw on the side with a large plastic knurled head. The antenna is secured by hand tightening the screw. There is a friction fit between the antenna mounting base and the flange. A white plastic cap fits onto the top of the flange when it is not in use and is held in place by the securing screw. The cap is secured to the flange with a 12 cm length of chain. The modem antenna mount can be seen in photographs one and two.

5.0 COMPUTER BOARDS

The single board computer (SBC) and input/output (I/O) board are mounted together. They are separated by four centimeters by four spacers mounted at the corners. The boards are supported vertically on a metal bracket. They are oriented so the air flow goes over the surfaces of the integrated circuits. The boards are parallel to the long side of the enclosure. They are mounted near the power amplifier and terminal strip. The ISA bus is connected using card edge connectors and ribbon cable. The TTL I/O signals that were added to the bus of the I/O card are hardwired to connectors directly using the ribbon cable. The computer boards can be seen in photographs eight, nine and ten.

6.0 INTERNAL COMPONENT LAYOUT

There have been some variations in the component layout from the description in the DSD. The main power supply board is mounted to the side using half inch spacers. It is on the side that has the former connector panel, but is now a blank panel. The secondary daughter board is mounted on the short side, directly beside the main power supply board, also on half inch spacers. It is between the fan and main power supply board, in the corner opposite from the switch/connector panel. The fan is oriented so it pulls air through the enclosure, rather than pushing it through. Figure one illustrates the internal component layout.

The RF components are located in a line, directly over the main heat sink. They use the heat sink for mounting by tapping the screw holes directly into the heat sink. The order of the components, from the fan to the switch/connector panel is; noise source, CTI synthesizer, ten megahertz oscillator, single sideband modulator (SSBM), RF switch, digital attenuator and power amplifier. The RF switch is mounted on the same mounting plate as the digital attenuator, but on the opposite side. The output amplifier is directly beside the switch/connector panel. The quadrature board is mounted in its old position, on the side wall, using half inch spacers. Photographs eight to ten show various aspects of the component layout.

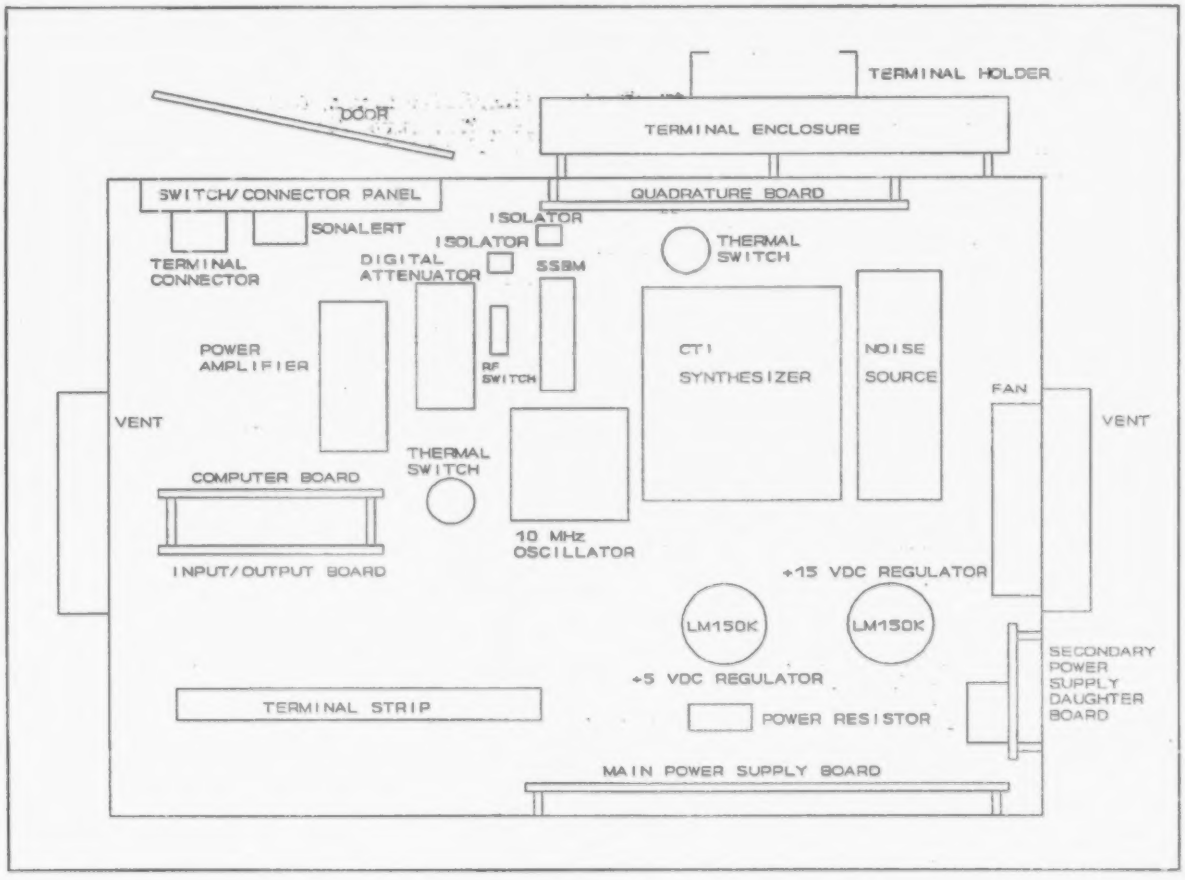


Figure One
PIFS Internal Component Layout

7.0 SOFTWARE

The software is essentially the same as the version presented in the DSD, with minor changes and additions. A hard copy of the latest version of the control software is shown in Appendix A. There were a few bugs that were fixed. The only functional change is in the third and fourth bytes of the status message. As shown below, the order of the bits has been changed so the least significant five bits are in the third byte while the eight most significant bits are in the fourth byte.

M_1 M_0 X D_4 D_3 D_2 D_1 D_0	D_{12} D_{11} D_{10} D_9 D_8 D_7 D_6 D_5
n_7 n_6 n_5 n_4 n_3 n_2 n_1 n_0	n_7 n_6 n_5 n_4 n_3 n_2 n_1 n_0
third byte of status	fourth byte of status

8.0 POWER SUPPLY BOARD CHANGES

The schematic diagram of the power supply daughter board was not included in the Design Specification Document. It is included in this document, in Appendix B. The updated schematic for the main power supply board is also shown in Appendix B.

There have been several minor changes made to the power supply board. The values of capacitors C1 and C6 have been changed to 22 µF. Zener diode ZR2 is a 1N4746A type, which is 18.0 volts. Transistors Q1 and Q2 have been changed to type 2N3904.

The under-voltage circuit has been changed. R21 has been replaced by a 5 Kohm potentiometer. It is adjusted to turn on the sonalert at a battery voltage of 18.0 volts. The op-amp U8B is not used and R23 has been removed. A jumper connects pin 1 of U10 to J4-13. These changes make the TTL level under-voltage signal to the computer an active low signal. The sonalert disable switch is connected in series between the sonalert and the five volt supply.

Some changes were made to the doppler frequency generation circuit. Resistor R19 has been removed. Resistor R18 has been changed to 820 ohms. Resistor R17 has been removed, so pin 15 of U7 is now unconnected.

9.0 TESTING

Testing was done in several stages, generally working from the component level up to the complete system level. The computer boards were tested with the supplied software first, and then with the control software. The control software was tested and made completely functional before the boards or controlled components were mounted inside the enclosure.

The main power supply board and its daughter board were individually bench tested with and without loads. The RF

components had been used in the local control PIFS, therefore there was no need to test them. The radio modems and their antennas were tested between two desktop computers at close range before mounting in the enclosures.

After all the components were mounted and wired inside the enclosures the individual component power supply lines were disconnected. Power was applied to the PIFS using a current limited bench power supply. After it was confirmed that the internal power supplies were working properly, each component was connected individually and checked. The coaxial cables were not connected between the RF components at this stage. Once all the components had power applied and checked, the coaxial cables were put on. Full local control was tested first using the hand-held terminal. Remote control was then checked by bypassing the modem and going directly to the computer board with a connection to a desktop PC. A program on the desktop PC, written in C++, was used to allow the operator to enter any command. After full control was obtained using the PC, the modems were connected and the PC was again used for control, via the modems.

9.1 Remote Control Tests

Remote control of the PIFS using the desktop PC was first exercised over a short distance. The PC was in building T-80 while the PIFS was in the tall bay, a distance of about one hundred meters. A spectrum analyzer and frequency counter were used to check the RF output. Once this test was successfully completed the PIFS was transported to the white trailer while the PC was moved into the tall bay. Full remote control was successful over this distance, about 1600 meters. The monopole antennas were used at each end of the control link. The larger standard gain horn was used on top of the PIFS, rather than the smaller horn that is shown in the photographs.

The final test involved taking the PIFS to Pinhey's Point, up the Ottawa River. This is a distance of almost seven kilometers. The PIFS was first set up on shore, on top of a van. This was not a direct line of sight since there were a lot of trees and possibly some land between the PIFS and the tall bay. The beacon receiver on the front end radar rack was used to receive the PIFS signal, which was observed on the spectrum analyzer. The exciter was set to provide the appropriate local oscillator to the beacon receiver. The amplitude of the received 60 MHz signal was barely visible over the noise on the spectrum analyzer. The PC did have complete remote control of all the PIFS functions, using the monopole antennas at each end of the link. A yagi antenna was also tried at one end and it also worked, as expected.

The PIFS was then moved down to the end of the dock at Pinhey's Point. It was setup about 30 cm off the water surface, in a direct line of sight with the tall bay. The received 60 MHz

signal increased by about 20 decibels over the previous position. All remote control functions worked using the monopole antennas. It was not possible to easily test the PIFS at any range over seven kilometers. This was because the key for the lock on the gate down to the river was lost. Therefore, it was very difficult to get the boat in the water. Also, the raft was high and dry on the shore.

9.2 Testing the Second PIFS

The first PIFS was completely built and then tested at Pinhey's Point before the second PIFS was completed. The second PIFS had been partially populated and wired, but not powered up. The testing procedures for the second PIFS were similar to the first PIFS, although it was not checked at a distant range. It was only checked under remote control within the radar test site compound. It was not necessary to check it at the distant range since it was known that the modem worked well. A problem was encountered while testing the second PIFS. The computer board would not boot up. After some troubleshooting it was determined that the board would have to be returned to Octagon for repair, under warranty. The computer board from the first PIFS was removed and put into the second PIFS for testing. The second PIFS was completely tested using the other computer board. The board should be returned from repair by the first week of September. This board was checked when received and it did work well.

9.3 Cooling Test

A cooling test was done to see how hot the inside of the enclosure gets. The PIFS was turned on and allowed to sit in the sun on a hot day, with an air temperature of 30 degrees Celsius and a light breeze. A thermometer probe was placed in the center of the inside of the PIFS. The internal air temperature rose slowly over a period of about one and a half hours to peak at about 48 degrees Celsius. The fan never came on since its thermal switches did not reach 50 degrees. The external heat sinks were warm to the touch, particularly the heat sink for the +5 VDC regulator. The fan was forced on by applying hot air from a heat gun to one of the thermal switches. The fan turned on and forced a slow movement of air through the enclosure. The temperature slowly dropped to about 46 degrees Celsius and remained at that value. Therefore, the fan never turned off. It is doubtful that the fan would come on at all, except on an extremely hot day.

9.4 Battery Load Test

A battery load test was done using fully charged batteries. The PIFS was started and allowed to run in operate mode, without changing any parameters. Unfortunately, the working day ran out before the batteries. After three and a half hours the battery voltage had dropped to 22.29 volts. From this test and the earlier load test it is estimated that the fully charged batteries should

last at least four hours with a steady operate condition. If the PIFS is switched to standby when not being used, the batteries should last at least five hours. When the PIFS was operated with a bench power supply, the current draw was measured on the supply meter. In operate mode the PIFS draws 4.5 amps on turn-on. After a few minutes the current goes down and remains at 4 amps. When switched to standby the current dropped to just 2 amps. Therefore, the PIFS will operate twice as long in the standby mode.

10.0 PHOTOGRAPHS

A series of photographs were taken of the first PIFS after it had been assembled and partially tested. Ten of these photographs have been selected for inclusion in this report and are presented after the conclusions. The second PIFS is similar in appearance, however the routing of the wiring is slightly neater. Each photograph shows a certain aspect of the PIFS. It is connected to the original battery box and has the modem monopole antenna mounted on it. The small standard gain horn is mounted on top, however the larger standard gain horn would normally be used in practice. Each photograph is described with a short paragraph.

Photograph One

An overall view of the PIFS is shown with the battery box in the background. The switch/connector panel cover is closed and the terminal is stored in its box. One of the support rails can be seen under the PIFS. Only the removal of the modem antenna is required to put the PIFS into the transportation configuration.

Photograph Two

An overall overhead view of the PIFS and battery box is shown. The hand-held terminal is placed in its operate position holder. The switch/connector door is open with the connections made for the batteries, RF output coaxial cable and terminal. The cover for the modem antenna mount can be seen hanging from its chain. The blank cover panel can be seen on the side beside the battery box. It covers the hole that was made for the previous connector panel.

Photograph Three

This shows the terminal in its holder, with the default power-up display. The terminal is connected to the switch/connector panel through the coiled cord. The terminal can be easily removed from or inserted into the holder.

Photograph Four

The terminal and its connection cord can be seen placed in the transportation position. The gasket and rubber padding material are on the inside of the open door. Rubber padding is also behind the terminal.

Photograph Five

This is a close-up detail of the switch/connector panel. The battery connection is on the lower left and the optional modem antenna N-type connector is in the upper left. The coaxial cable for the horn is connected to the SMA connector, labelled "RF OUTPUT". Directly below it the black sonalert can be seen. The modular terminal connector is on the lower right of the panel and the three toggle switches are above it. The side of the main external heat sink is visible under the enclosure.

Photograph Six

The three heat sinks are shown on the underside of the enclosure. The paint has been taken off the entire underside. The two TO-3 package voltage regulators are mounted on the two black heat sinks. The support rails are visible at the top and bottom of the photograph. The switch/connector panel is at the top.

Photograph Seven

This is a view of the underside of the lid of the enclosure, showing how the modem is mounted. The hole for the antenna coaxial cable is in the lower left corner of the lid. The three screws which secure the flange can be seen around the hole. The monopole antenna is visible in the background on the far left of the photograph. It is black, with a silver coloured cap and an orange band near the top.

Photograph Eight

This is a general overhead view of the inside of the enclosure. It is oriented the same way as Figure One. The switch/connector panel is in the upper left. The computer boards are on the left and the main power supply board is in the lower right. The chain to hold the lid in the open position is on the right.

Photograph Nine

This shows an oblique view of the enclosure interior. The switch/connector panel is on the left, the computer boards are in the foreground and the fan is in the right background.

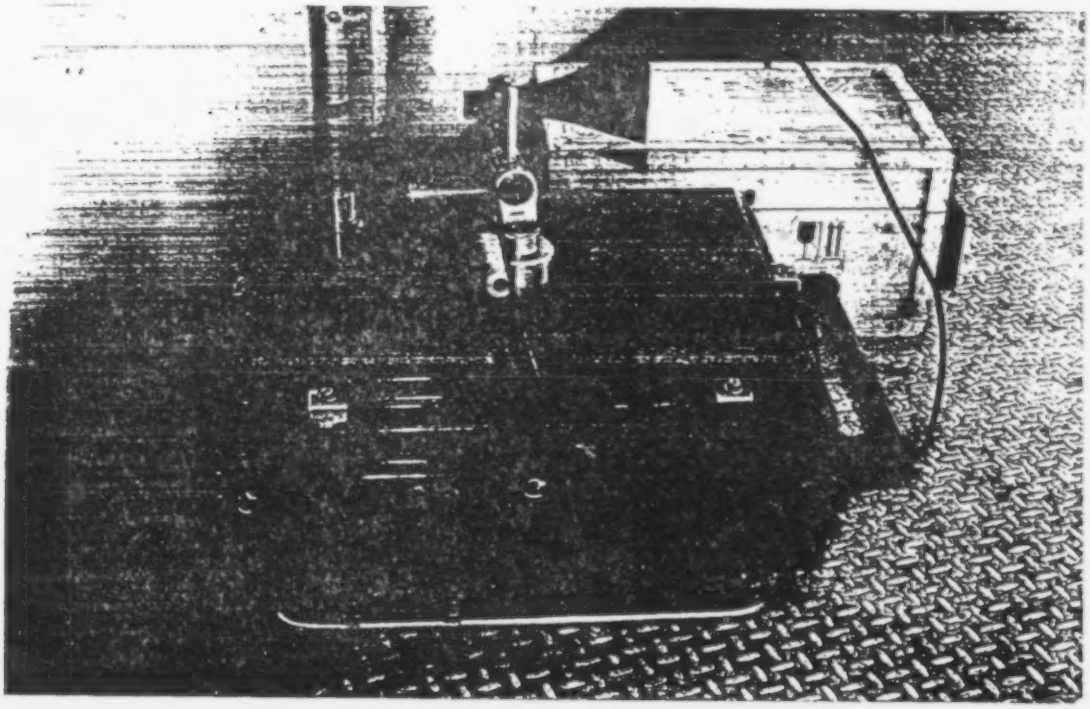
Photograph Ten

This is another oblique view of the interior, taken in the opposite direction from photograph nine. The main power supply board is on the left rear wall and the CTI microwave source can be seen in the foreground. The switch/connector panel is on the lower right.

11.0 CONCLUSIONS

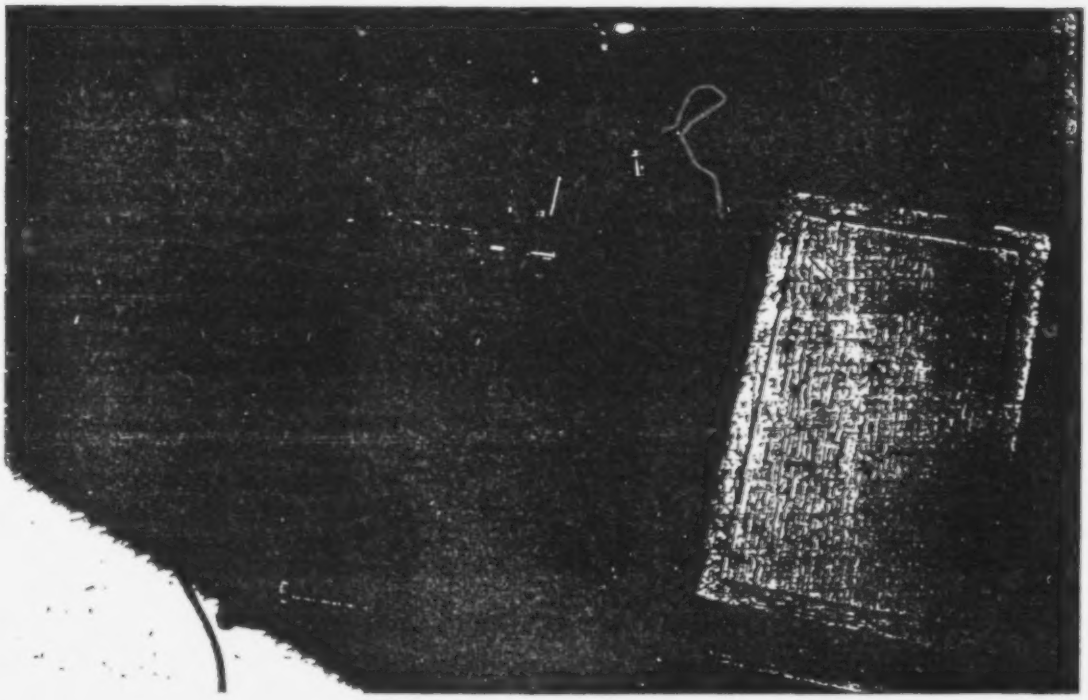
The changes made to the local control PIFS have produced two successful, functional remote controlled PIFS. They exceed the requirements of the statement of work and will be very useful for use with the EARS. An extensive market survey was done two years ago to find a low cost jammer that could be used with the EARS system. At the time, the cheapest device available that fulfilled the requirements cost over $200,000. This was considered inexpensive by industry standards. It was decided after this survey that jammers must be custom-made and the remote controlled PIFS is the result.

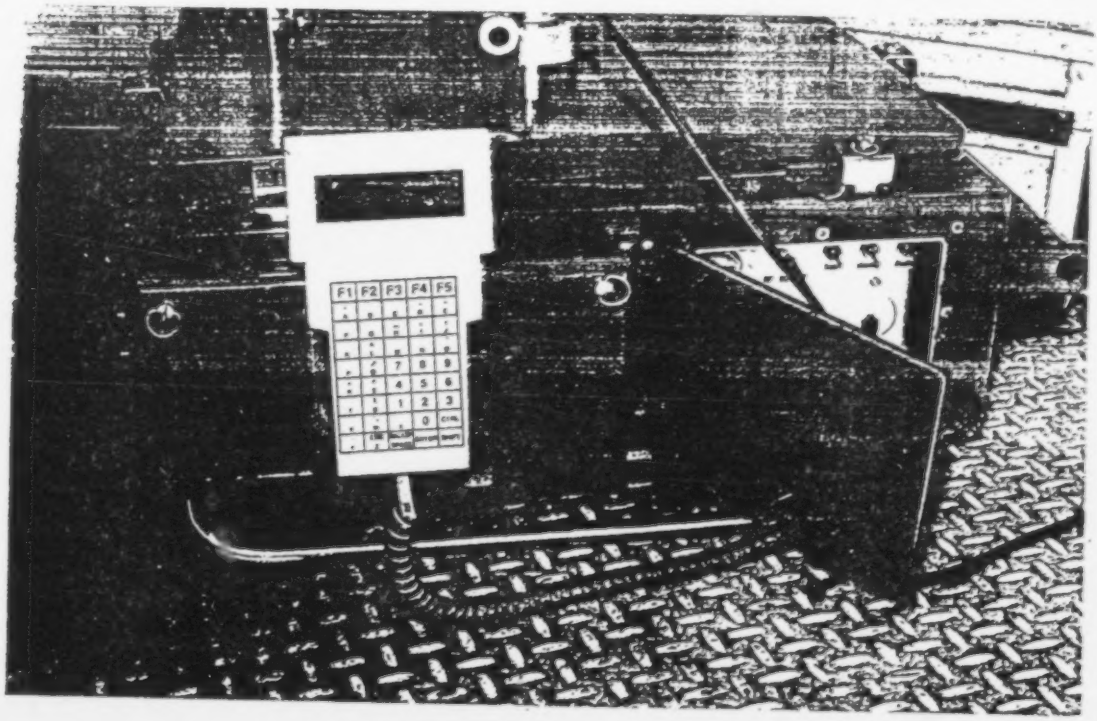
The only major drawback with the PIFS is the speed of the control. The radio modem link is very slow, taking around 30 milliseconds to communicate one or two command bytes. The CTI synthesizer is also very slow, taking up to 50 milliseconds to change frequencies. It is possible to use the EARS transmitter itself to communicate the command words. This would be much faster, taking perhaps 10 microseconds to communicate the command words. Further development would be required to enable this. It would also be possible to purchase synthesizers that can change frequency much faster. If the PIFS were to be made again, there could be many minor design improvements done. It could be made smaller, lighter, faster and more efficient.



PHOTOGRAPH ONE
Overall Oblique View With Battery Box

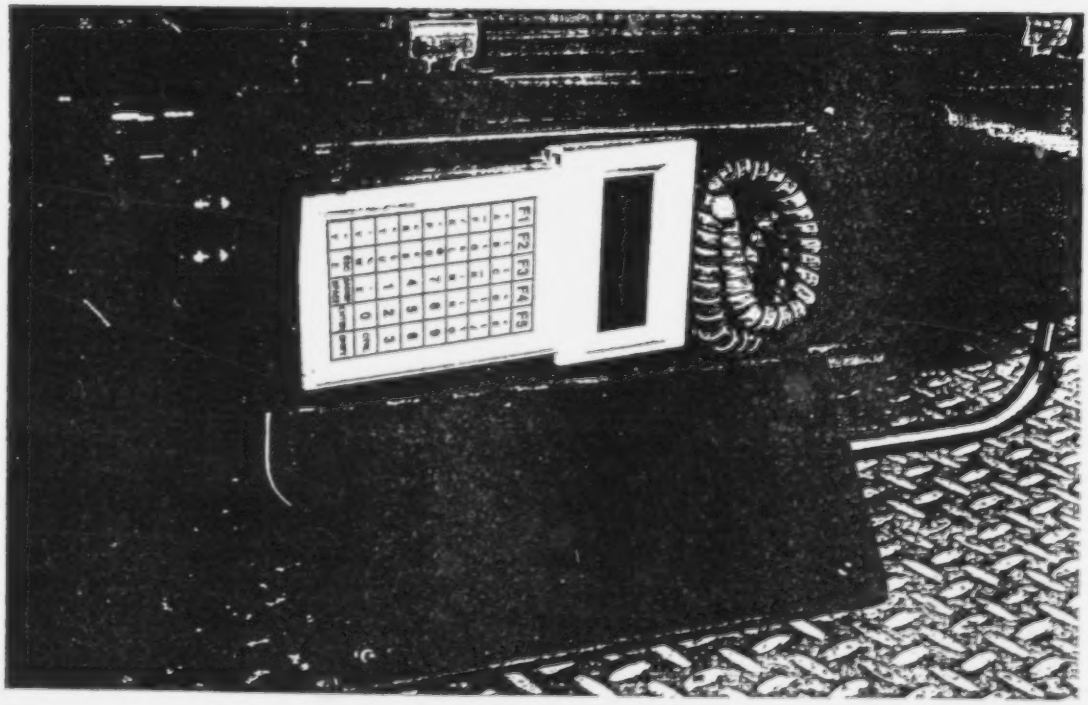
PHOTOGRAPH TWO
Overall Overhead View With Battery Box

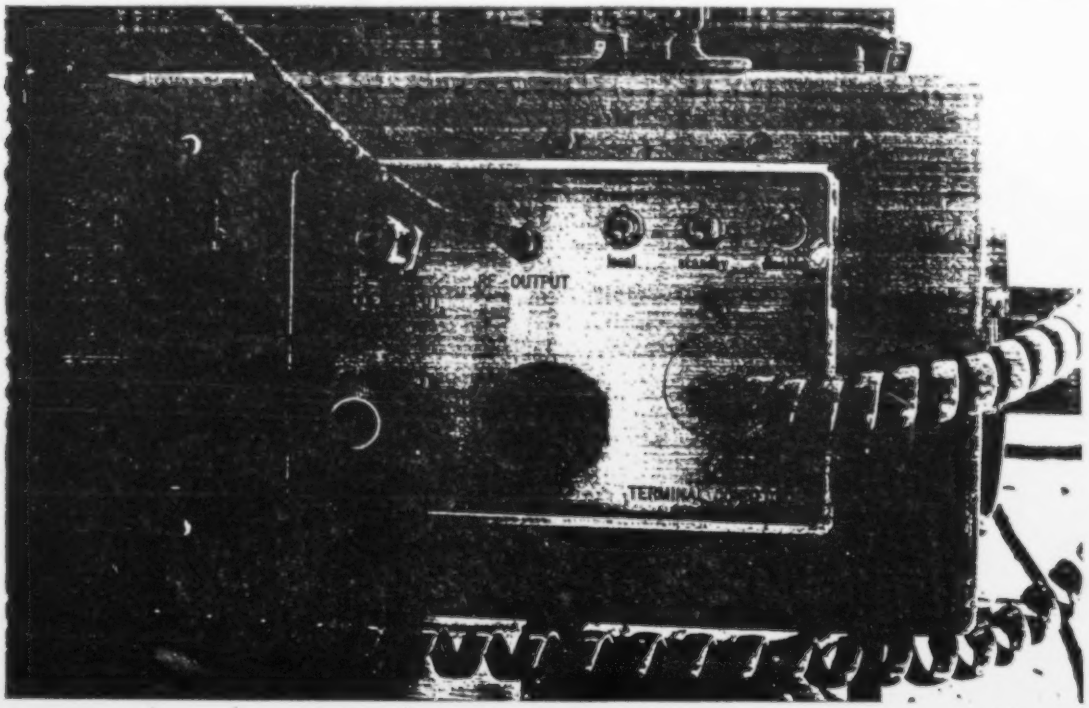




PHOTOGRAPH THREE
Hand-Held Terminal in Holder

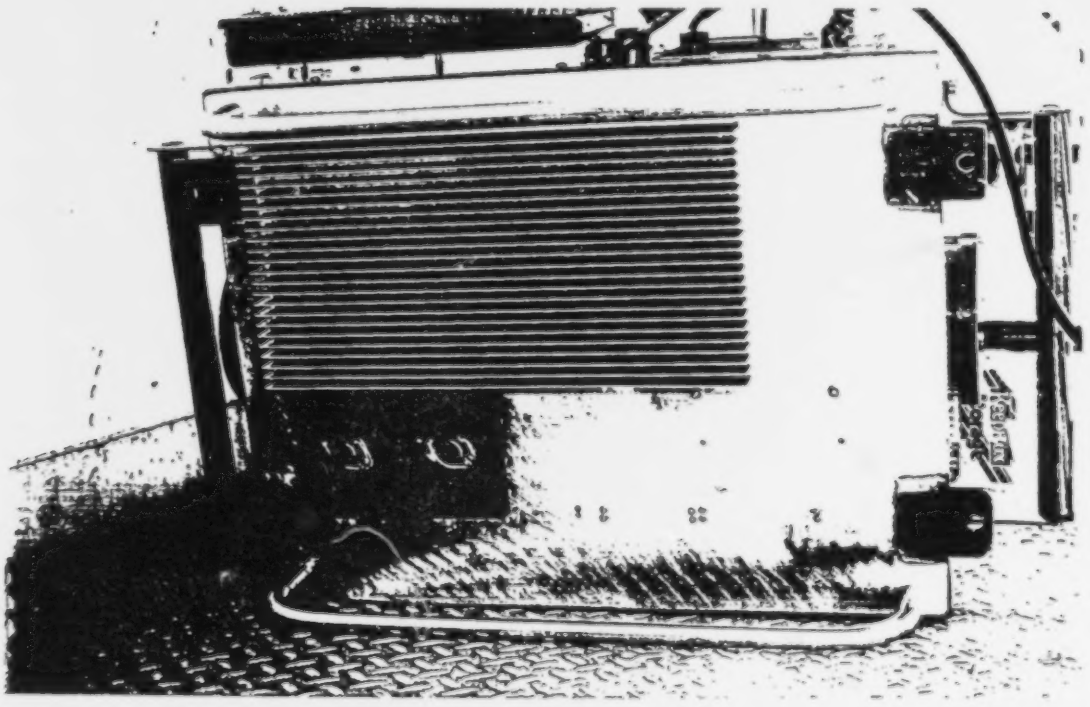
PHOTOGRAPH FOUR
Hand-Held Terminal and Cord in Storage Box

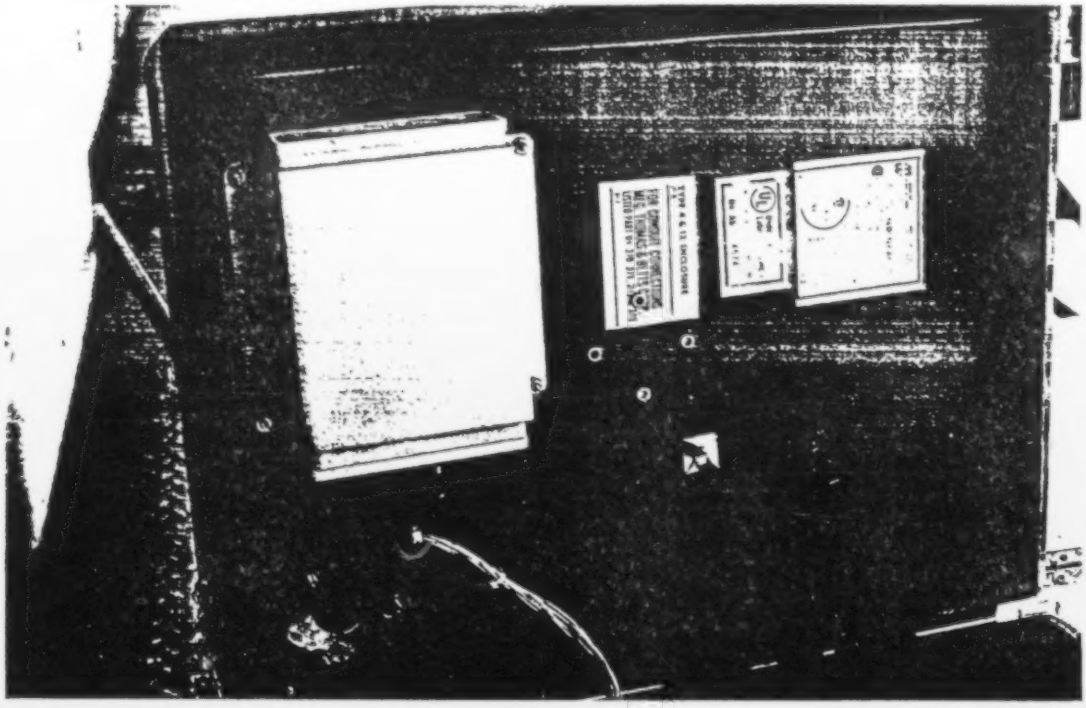




PHOTOGRAPH FIVE
Detail of Switch/Connector Panel

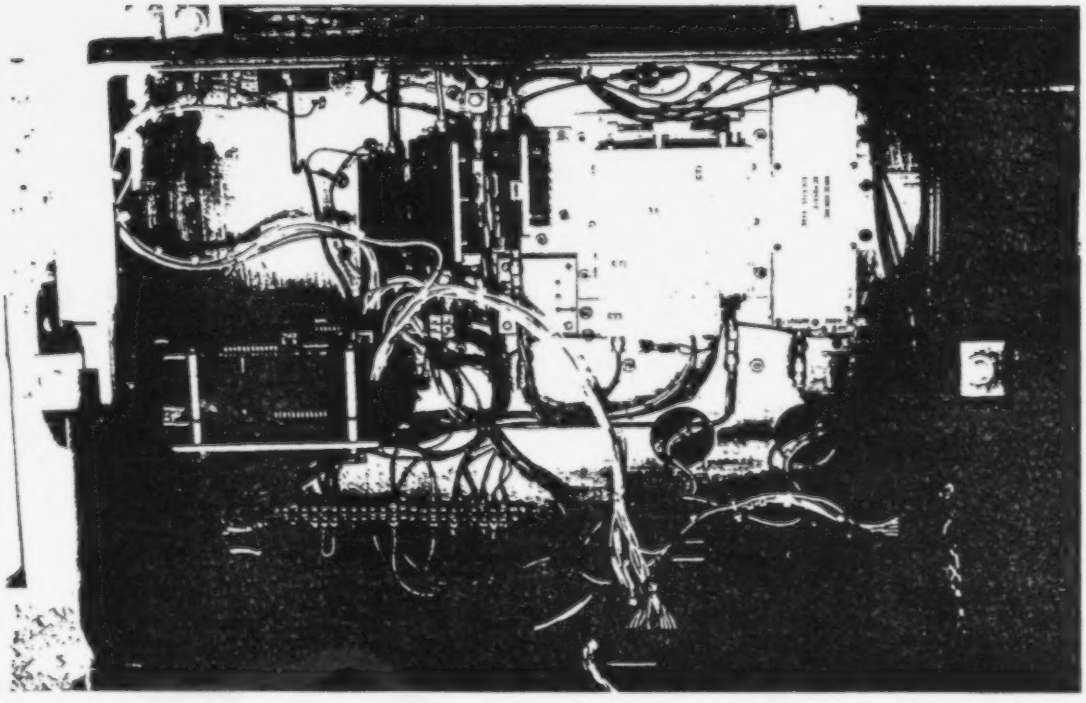
PHOTOGRAPH SIX
Underside With Heat Sinks

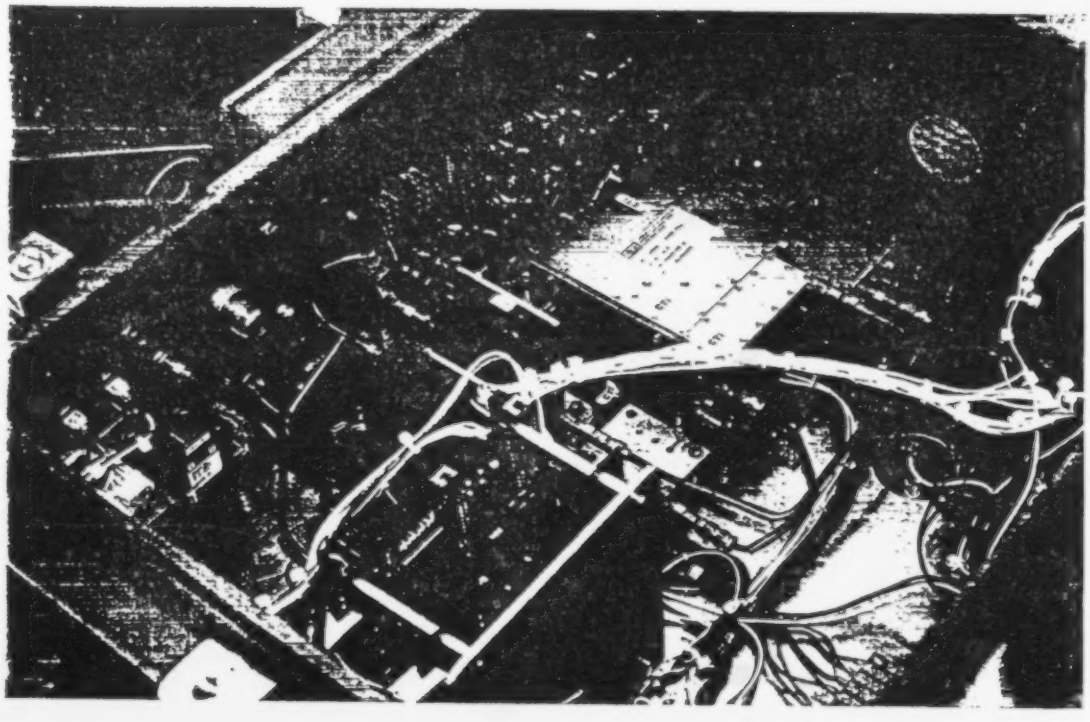




PHOTOGRAPH SEVEN
Modem Mounting and Hole for Antenna Connector

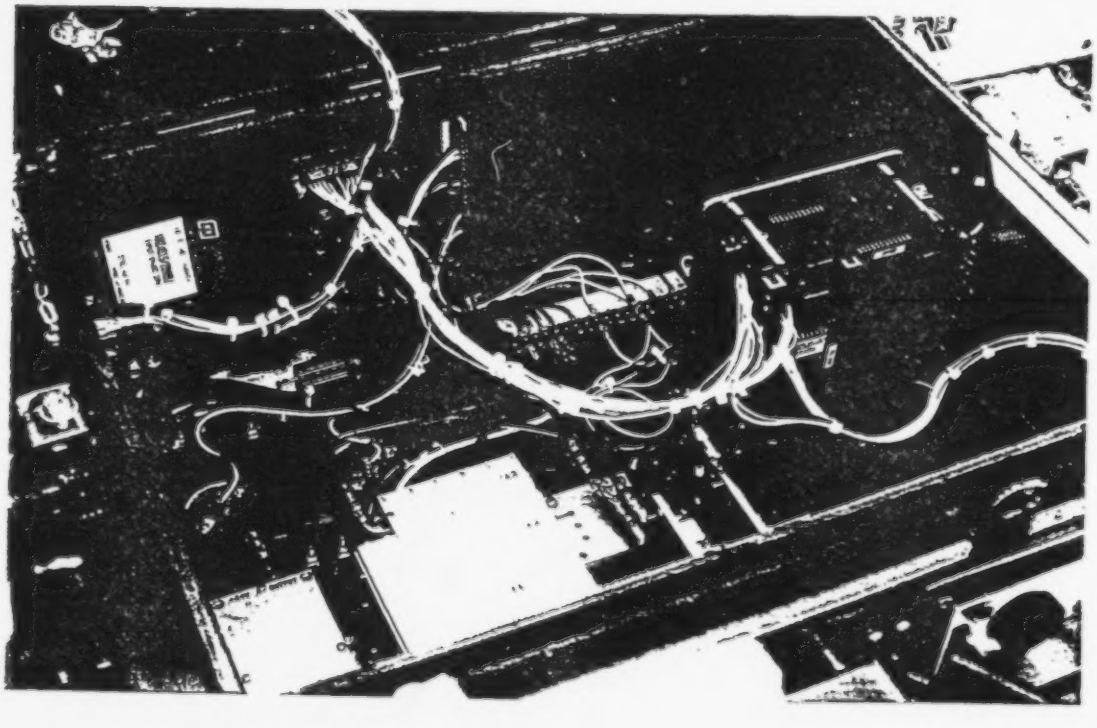
PHOTOGRAPH EIGHT
Overhead View of Interior





PHOTOGRAPH NINE
Interior Showing Panel, CTI Source and Fan

PHOTOGRAPH TEN
Interior Showing Main Power Board and Computer



APPENDIX A

CONTROL PROGRAM LISTING

```
/*This program will control the components in the Portable Interference
  Frequency Source.  It will be downloaded in Octagon's single board
  computer's flash non-volatile memory.*/


#include "async2.h"
#include <conio.h>
#include <bios.h>
#include <stdio.h>
#include <string.h>
#include <dos.h>
#include <stdlib.h>


#define BASE_ADDR   0x100
#define _8MHz       8000000.0

void init_comports(void);
void set_IOports(void);
void send_defaults(void);
void init_terminal(void);
void CTI_loop(void);
void status_display(void);
void test_local_remote_flag(void);
void test_alarms(void);
void local_mode(void);
void remote_mode(void);
void send_ready(void);
void send_local(void);
void local_control(void);
void get_char_remote(void);
void get_char(void);
void get_char2(void);
void interpret_remote_command(void);
void define_command(void);
void set_RF_frequency(void);
void set_attenuation(void);
void set_RF_enable(void);
void set_operate_standby(void);
void set_operate_standby_2(void);
void set_operate_standby_3(void);
void set_modulation(void);
void send_status(void);
void set_doppler_frequency(void);



int loc_rem_bit, loc_rem_test;
int flagtest, temp_int, flag = 0;
int attenuation_int = 0, RF_freq_int = 8900;
int transmit_enable = 1, err, modulation=1;
int byte_1 = 0, byte_2 = 0, byte_3 = 0, byte_4 = 0;
int least_5 = 0, most_8 = 0;
char remote_command, local_command;
```

```
char RF_freq_ascii[4], temp_char1[2], temp_char2[3], temp_char3[4],
temp_char4[5];
char attenuation_char[3], doppler_frequency_char[6], receive_ch, character[2];
unsigned int command = 0, RF_frequency = 0, atten_RF_en = 1, op_st_dc_dn = 1;
unsigned long int command_temp, temp2_int;
unsigned long int doppler_frequency_int = 0;
unsigned long int counter_0_value_int, counter_1_value_int;
unsigned long int counter_clock, filter_clock;
unsigned long int counter_0_value, counter_1_value;


main()
{
  init_comports();
  set_IOports();
  send_defaults();
  init_terminal();
  CTI_loop();
  status_display();
  test_local_remote_flag();
  test_alarms();
  return(0);
}


void init_comports(void)
{
  sethandshakemode(0,0);
              //set port1 no handshaking
  sethandshakemode(1,0);                  //set port2 no handshaking
  opencom(0,9600,0,1,8,0);                //set port1 9600,8,N,1
  opencom(1,9600,0,1,8,0);                //set port2 9600,8,N,1
  return;
}


void set_IOports(void)
{                                         //set port A as output
  outp(BASE_ADDR + 0x03, 0x88);           //set port B as output
  return;                                 //set port C MSB as output
}                                         //set port C LSB as input


void send_defaults(void)
{
  outp(BASE_ADDR + 0x00, RF_frequency); //send the default parameters
  outp(BASE_ADDR + 0x01, atten_RF_en);  //to the IO lines
  outp(BASE_ADDR + 0x08, op_st_dc_dn);
  return;
}
```

```
void init_terminal(void)
{
  writecom(1,"\x1B[?8l",&err);  // disable key repeat
  writecom(1,"\x1B[0t",&err);   // enable key click
  writecom(1,"\x1B[3t",&err);   // disable Knp function
  writecom(1,"\x1B[4t",&err);   // enable cursor
  writecom(1,"\x1B[6t",&err);   // enable cursor blink
  writecom(1,"\x1B[5zf1;52\x1B[0z",&err);  // F1 = 'R' for RF ON/OFF
  delay(100);          // allow time for the 8045 to program the function key
  writecom(1,"\x1B[5zf2;46\x1B[0z",&err);  // F2 = 'F' for RF frequency
  delay(100);
  writecom(1,"\x1B[5zf3;41\x1B[0z",&err);  // F3 = 'A' for attenuation
  delay(100);
  writecom(1,"\x1B[5zf4;54\x1B[0z",&err);  // F4 = 'T' for modulation type
  delay(100);
  writecom(1,"\x1B[5zf5;44\x1B[0z",&err);  // F5 = 'D' for doppler frequency
  delay(75);
  return;
}


void CTI_loop(void)
{
  do
    {
     loc_rem_bit = inp(BASE_ADDR + 0x02);
    }
  while(loc_rem_bit & 0x20);
  return;
}


void status_display(void)
{
  writecom(1,"\x1B[1s",&err);  // erase entire display and send cursor home
  writecom(1,"TX",&err);
  if (transmit_enable == 1) writecom(1," ON      ",&err);
  else writecom(1," OFF     ",&err);
  itoa(RF_freq_int, RF_freq_ascii, 10);
  strncpy(temp_char1, RF_freq_ascii, 1);
  writecom(1,temp_char1,&err);              // write the MHz thousands value
  writecom(1,".",&err);
  strncpy(temp_char2, RF_freq_ascii, 2);  // copy the first two digits
  strrev(temp_char2);                       // reverse the string
  strncpy(temp_char1, temp_char2, 1);
  writecom(1,temp_char1,&err);              // write the MHz hundreds value
  strncpy(temp_char3, RF_freq_ascii, 3);  // copy the first two digits
  strrev(temp_char3);                       // reverse the string
  strncpy(temp_char1, temp_char3, 1);
  writecom(1,temp_char1,&err);              // write the MHz tens value
```

```
  strncpy(temp_char4, RF_freq_ascii, 4);  // copy the first two digits
  strrev(temp_char4);                     // reverse the string
  strncpy(temp_char1, temp_char4, 1);
  writecom(1,temp_char1,&err);            // write the MHz tens value
  writecom(1," GHz",&err);
  writecom(1,"ATT=",&err);
  itoa(attenuation_int, attenuation_char, 10);
  if (attenuation_int > 9) writecom(1,attenuation_char,&err);
  else
  {
    writecom(1," ",&err);
    writecom(1,attenuation_char,&err);
  }
  writecom(1," dB ",&err);
  switch (modulation)
  {
    case 1: {
      writecom(1,"  NO MODUL",&err);
      break;
    }
    case 2: {
      writecom(1,"DOPPLR MOD",&err);
      break;
    }
    case 3: {
      writecom(1," NOISE MOD",&err);
      break;
    }
  }
  writecom(1,"DOPP= ",&err);
  itoa(doppler_frequency_int,doppler_frequency_char,10);
  if (doppler_frequency_int < 10000) writecom(1," ",&err);
  if (doppler_frequency_int < 1000) writecom(1," ",&err);
  if (doppler_frequency_int < 100) writecom(1," ",&err);
  if (doppler_frequency_int < 10) writecom(1," ",&err);
  writecom(1,doppler_frequency_char,&err);
  writecom(1," Hz",&err);
  writecom(1,"\x1B[4;1f",&err);  // position cursor to start of last row
  writecom(1,"M=FUNCTION KEY MENU",&err);
  delay(100);
  return;
}


void menu_display(void)
{
  writecom(1,"\x1B[1s",&err);  // erase entire display and send cursor home
  writecom(1,"F1=RF OUTPUT ON/OFF ",&err);
  writecom(1,"F2=RF FREQUENCY     ",&err);
  writecom(1,"F3=ATTEN. F4=MODULA.",&err);
  writecom(1,"F5=DOPPLER FREQ.",&err);
  delay(2000);        // wait for four seconds
  return;
```

```
}


void get_digit(void)
{
  get_char2();
  character[0] = receive_ch;
  temp_int = atoi(character);
  return;
}


void get_char(void) // get a character from the com port until the buffer is
empty
{
  checkcom(1,&receive_ch,&err);
  if(err != 0)
  {test_alarms();}
  return;
}


void get_char2(void)
{
  do
    {
     checkcom(1,&receive_ch,&err);
    }
  while(err != 0);
  return;
}


void get_char_remote(void)
{
  checkcom(0,&remote_command,&err);
  if(err != 0)
  {test_alarms();}
  interpret_remote_command();
  return;
}


void test_local_remote_flag(void)
{
  loc_rem_bit = inp(BASE_ADDR + 0x02);  //input the bit for local or
  if(loc_rem_bit & 0x40)                //remote status
  {flag = 1;}                           //1 = local
  flag = 0;                             //0 = remote
```

```
  return;
}


void test_alarms(void)
{
  loc_rem_bit = inp(BASE_ADDR + 0x02);
  if(loc_rem_bit & 0x20)
  {
    char message3 [ ] = "A";
    writecom(0,message3,0);
    command = 167;
    set_operate_standby();
  }

  if((loc_rem_bit & 0x80) == 0)
  {
    char message4 [ ] = "U";
    writecom(0,message4,0);
    command = 167;
    set_operate_standby();
  }

  flagtest = (loc_rem_bit & 0x40);
  flagtest = flagtest >> 6;
  if(flagtest == flag)
              {if(loc_rem_bit & 0x40)
                    {
                    local_mode();
                    }
                    remote_mode();
              }

  if(flagtest != flag)
              {delay(500);
               if(loc_rem_bit & 0x40)
                    {
                    local_mode();
                    }
                    remote_mode();
              }

  return;
}


void local_mode(void)
{
  if(flag == 0)
  {
    send_local();
    flag = 1;
```

```
    }

    if(loc_rem_bit & 0x10)
    {
      command = 167;
      set_operate_standby_2();
    }

    if((loc_rem_bit & 0x10) == 0)
    {
      command = 168;
      set_operate_standby_3();
    }

    return;
}


void remote_mode(void)
{
    if(flag == 1)
    {
      send_ready();
      flag = 0;
    }
    get_char_remote();
    return;
}


void send_ready(void)
{
    char message1 [ ] = "R";
    writecom(0,message1,0);
    return;
}


void send_local(void)
{
    char message2 [ ] = "L";
    writecom(0,message2,0);
    return;
}


void local_control(void)
      {

      get_char();                    // wait for character from 8045 terminal
```

```
      switch (receive_ch)
      {
        case 'M': {             // M means to display the menu for a short time
        menu_display();
        status_display();
        break;
        }
        case 'R': {             // R toggles the RF output on/off
        writecom(1,"\x1B[1;4H",&err);  // position cursor
        if (transmit_enable == 0)
        {
          writecom(1," ON ",&err);
          transmit_enable = 1;
          command = 0xA5;
        }
        else
        {
          writecom(1," OFF",&err);
          transmit_enable = 0;
          command = 0xA6;
        }

        writecom(1,"\x1B[4;20H",&err);  // position cursor to lower right
          set_RF_enable();
          break;
        }
        case 'F': {                          // F will set the RF frequency
        writecom(1,"\x1B[1;12H",&err);  // position cursor on GHz ones
        do
        {
          get_digit();                       // wait for GHz ones input
          if (temp_int < 8) writecom(1,"\x1B[0q",&err);  // sound short bell
        } while ( temp_int < 8 );
        RF_freq_int = 1000*temp_int;
        writecom(1,character,&err);
        writecom(1,".",&err);
        do
        {
          get_digit();                       // wait for MHz hundreds input
        if (((RF_freq_int == 9000) && (temp_int > 4)) || ((RF_freq_int == 8000)
&& (temp_int != 9))) writecom(1,"\x1B[0q",&err);  // sound short bell
        } while (((RF_freq_int == 9000) && (temp_int > 4)) || ((RF_freq_int ==
8000) && (temp_int != 9)));
        RF_freq_int = RF_freq_int + 100*temp_int;
        writecom(1,character,&err);
        do
        {
          get_digit();                       // wait for MHz tens input
        if ((RF_freq_int == 9400) && (temp_int > 0)) writecom(1,"\x1B[0q",&err);
// sound short bell
        } while ((RF_freq_int == 9400) && (temp_int > 0));
        RF_freq_int = RF_freq_int + 10*temp_int;
        writecom(1,character,&err);
        do
```

```
      {
        get_digit();                          // wait for MHz ones input
      if (((temp_int != 0) && (temp_int != 5)) || ((RF_freq_int == 9400) &&
(temp_int != 0))) writecom(1,"\x1B[0q",&err);  // sound short bell
      } while (((temp_int != 0) && (temp_int != 5)) || ((RF_freq_int == 9400)
&& (temp_int != 0)));
      RF_freq_int = RF_freq_int + temp_int;
      writecom(1,character,&err);
      command = ((RF_freq_int - 8900)/5);

      writecom(1,"\x1B[4;20H",&err);  // position cursor to lower right
        set_RF_frequency();
        break;
      }
      case 'A': {             // A sets the attenuation
      writecom(1,"\x1B[2;5H",&err);  // position cursor on attenuation tens
      do
      {
        get_digit();                          // wait for attenuation tens input
      if ( temp_int > 6 ) writecom(1,"\x1B[0q",&err);  // sound short bell
      } while ( temp_int < 0 || temp_int > 6 );
      attenuation_int = 10 * temp_int;
      if (temp_int == 0)
      {
        writecom(1," ",&err);           // put in blank space instead of '0'
      }
      else
      {
        writecom(1,character,&err);
      }
      do
      {
        get_digit();                          // wait for attenuation ones input
      if ( (temp_int > 3) && (attenuation_int == 60) || temp_int < 0 ||
temp_int > 9) writecom(1,"\x1B[0q",&err);  // sound short bell
      } while ( (temp_int > 3) && (attenuation_int == 60) );
      writecom(1,character,&err);
      attenuation_int = attenuation_int + temp_int;
      command = (attenuation_int + 101);

      writecom(1,"\x1B[4;20H",&err);  // position cursor to lower right
        set_attenuation();
        break;
      }
      case 'T': {             // T sets the modulation type
      writecom(1,"\x1B[2;11H",&err);  // position cursor on modulation type
      modulation++;                   // increment modulation type
      if (modulation == 4) modulation = 1;
      switch (modulation)             // change modulation type display
      {
        case 1: {
        writecom(1,"  NO MODUL",&err);
        break;
        }
```

```
          case 2: {
          writecom(1,"NOISE MOD",&err);
          break;
          }
          case 3: {
          writecom(1," DOPPLR MOD",&err);
          break;
          }

         }
         command = (modulation + 168);
           writecom(1,"\x1B[4;20H",&err);  // position cursor to lower right
         set_modulation();
           break;
        }

         case 'D': {            // D is for setting the doppler frequency
        writecom(1,"\x1B[3;8H",&err);  // position cursor on doppler frequency
        do
        {
          get_digit();                  // wait for doppler 1,000 input
          if (temp_int > 7)
            writecom(1,"\x1B[0q",&err);  // sound short bell
        } while (temp_int > 7);
        if ( temp_int == 0)
          writecom(1," ",&err);  // put in blank space
        else writecom(1,character,&err);
        doppler_frequency_int = 1000 * temp_int;
        do
        {
          get_digit();                  // wait for doppler hundreds input
          if (doppler_frequency_int == 7000 && temp_int == 9)
            writecom(1,"\x1B[0q",&err);  // sound short bell
        } while (doppler_frequency_int == 7000 && temp_int == 9);
        if ( temp_int == 0 && doppler_frequency_int == 0) writecom(1,"_",&err);
// put in blank space
        else writecom(1,character,&err);
        doppler_frequency_int = doppler_frequency_int + 100 * temp_int;
        do
        {
          get_digit();                  // wait for doppler tens input
          if (doppler_frequency_int == 7800 && temp_int > 2)
            writecom(1,"\x1B[0q",&err);  // sound short bell
        } while (doppler_frequency_int == 7800 && temp_int > 2);
        if ( temp_int == 0 && doppler_frequency_int == 0) writecom(1," ",&err);
// put in blank space
        else writecom(1,character,&err);
        doppler_frequency_int = doppler_frequency_int + 10 * temp_int;
        do
        {
          get_digit();                  // wait for doppler ones input
          if (doppler_frequency_int == 7810 && temp_int > 3)
            writecom(1,"\x1B[0q",&err);  // sound short bell
        } while (doppler_frequency_int == 7810 && temp_int > 3);
```

```
        writecom(1,character,&err);
        writecom(1," Hz      ",&err);
        doppler_frequency_int = doppler_frequency_int + temp_int;

        writecom(1,"\x1B[4;20H",&err);  // position cursor to lower right
          set_doppler_frequency();
          break;
        }
          }
          test_alarms();
          return;
          }


void interpret_remote_command(void)              //this section converts the
{                                                //ASCII character received
       command = 0;                             //over the serial port to an
                                                //integer one bit at a time
       if(remote_command & 0x80)
            {command = (command | 0x80);}

       if(remote_command & 0x40)
            {command = (command | 0x40);}

       if(remote_command & 0x20)
            {command = (command | 0x20);}

       if(remote_command & 0x10)
            {command = (command | 0x10);}

       if(remote_command & 0x08)
            {command = (command | 0x08);}

       if(remote_command & 0x04)
            {command = (command | 0x04);}

       if(remote_command & 0x02)
            {command = (command | 0x02);}

       if(remote_command & 0x01)
            {command = (command | 0x01);}

      define_command();
        return;
      }


void define_command(void)                        //this section will goto the
      {                                          //function defined by the
      if(command <= 0x64)                        //remote command
            {set_RF_frequency();}
```

```
if(0x65 <= command && command <= 0xA4)
      {set_attenuation();}

if(0xA5 <= command && command <= 0xA6)
      {set_RF_enable();}

if(0xA7 <= command && command <= 0xA8)
      {set_operate_standby();}

if(0xA9 <= command && command <= 0xAB)
      {set_modulation();}

if(command == 0xAC)
      {send_status();}

if(command >= 0xE0)
      {
            command = (command & 0x1F);
            command = least_5;
            do
            {
            checkcom(0,&receive_ch,&err);
            } while(err != 0);
            temp2_int = 0;       //reset the temporary integer

   if(receive_ch & 0x80)
      {temp2_int = (temp2_int | 0x80);}

 if(receive_ch & 0x40)
      {temp2_int = (temp2_int | 0x40);}

 if(receive_ch & 0x20)
      {temp2_int = (temp2_int | 0x20);}

 if(receive_ch & 0x10)
      {temp2_int = (temp2_int | 0x10);}

 if(receive_ch & 0x08)
      {temp2_int = (temp2_int | 0x08);}

 if(receive_ch & 0x04)
      {temp2_int = (temp2_int | 0x04);}

 if(receive_ch & 0x02)
      {temp2_int = (temp2_int | 0x02);}

 if(receive_ch & 0x01)
      {temp2_int = (temp2_int | 0x01);}

            temp2_int = most_8;
            temp2_int = (temp2_int << 5);
            doppler_frequency_int = 0;      //reset the doppler frequency
            doppler_frequency_int = (temp2_int | command);
            set_doppler_frequency();
```

```
                }

        test_alarms();
          return;
        }


void set_RF_frequency(void)
     {
     RF_frequency = 0;
     command = (command * 5);

     if(command >= 400)
        {
        RF_frequency = (RF_frequency | 0x80);
        command = (command - 400);
        }

     if(command >= 200)
        {
        RF_frequency = (RF_frequency | 0x40);
        command = (command - 200);
        }

     if(command >= 100)
        {
        RF_frequency = (RF_frequency | 0x20);
        command = (command - 100);
        }

     if(command >= 80)
        {
        RF_frequency = (RF_frequency | 0x10);
        command = (command - 80);
        }

     if(command >= 40)
        {
        RF_frequency = (RF_frequency | 0x08);
        command = (command - 40);
        }

     if(command >= 20)
        {
        RF_frequency = (RF_frequency | 0x04);
        command = (command - 20);
        }

     if(command >= 10)
        {
        RF_frequency = (RF_frequency | 0x02);
        command = (command - 10);
```

```
        }

    if(command >= 5)
        {
        RF_frequency = (RF_frequency | 0x01);
        command = (command - 5);
        }

    outp(BASE_ADDR + 0x00, RF_frequency);       //set the output lines to
    test_alarms();                              //the desired RF_frequency
    return;
    }


void set_attenuation(void)
    {
     unsigned int attenuation = 0;
     command = (command - 0x65);

     if(command >= 32)
       {
        attenuation = (attenuation | 0x40);
        command = (command - 32);
       }

     if(command >= 16)
       {
        attenuation = (attenuation | 0x20);
        command = (command - 16);
       }

     if(command >= 8)
       {
        attenuation = (attenuation | 0x10);
        command = (command - 8);
       }

     if(command >= 4)
       {
        attenuation = (attenuation | 0x08);
        command = (command - 4);
       }

     if(command >= 2)
       {
        attenuation = (attenuation | 0x04);
        command = (command - 2);
       }

     if(command >= 1)
       {
        attenuation = (attenuation | 0x02);
        command = (command - 1);
```

```
        }

      atten_RF_en = (atten_RF_en & 0x01);
      atten_RF_en = (atten_RF_en | attenuation);
      outp(BASE_ADDR + 0x01, atten_RF_en);        //set the output lines to
      test_alarms();                              //the desired attenuation
      return;
      }


void set_RF_enable(void)
     {

     if(command == 0xA5)
        {
        atten_RF_en = (atten_RF_en | 0x01);
          }

     if(command == 0xA6)
        {
        atten_RF_en = (atten_RF_en & 0xFE);
          }

     outp(BASE_ADDR + 0x01, atten_RF_en);         //set the RF_enable switch
     test_alarms();                               //to on or off
     return;
     }


void set_operate_standby(void)
     {

      if(command == 0xA7)
        {
        op_st_dc_dn = (op_st_dc_dn & 0xF3);
        op_st_dc_dn = (op_st_dc_dn | 0x04);
          }

      if(command == 0xA8)
        {
        op_st_dc_dn = (op_st_dc_dn & 0xF3);
        op_st_dc_dn = (op_st_dc_dn | 0x08);
          }

     outp(BASE_ADDR + 0x02, op_st_dc_dn);         //set mode to operate or
     //op_st_dc_dn = (op_st_dc_dn & 0xF3);
     //delay(500);
     //outp(BASE_ADDR + 0x02, op_st_dc_dn);
     //status_display();
     CTI_loop();                                  //standby
     test_alarms();
     return;
```

```
}


void set_operate_standby_2(void)
    {
      op_st_dc_dn = (op_st_dc_dn & 0xF3);
      op_st_dc_dn = (op_st_dc_dn | 0x04);
      outp(BASE_ADDR + 0x02, op_st_dc_dn);      //set mode to stanby
      //op_st_dc_dn = (op_st_dc_dn & 0xF3);
      //delay(500);
      //outp(BASE_ADDR + 0x02, op_st_dc_dn);
      //status_display();
      test_alarms();
      return;
    }


void set_operate_standby_3(void)
    {
      op_st_dc_dn = (op_st_dc_dn & 0xF3);
      op_st_dc_dn = (op_st_dc_dn | 0x08);
      outp(BASE_ADDR + 0x02, op_st_dc_dn);      //set mode to operate
      //op_st_dc_dn = (op_st_dc_dn & 0xF3);
      //delay(500);
      //outp(BASE_ADDR + 0x02, op_st_dc_dn);
      //status_display();
      CTI_loop();
      local_control();
      return;
    }


void set_modulation(void)
    {

     if(command == 0xA9)
       {
       op_st_dc_dn = (op_st_dc_dn & 0xFC);
       }

     if(command == 0xAA)
       {
       op_st_dc_dn = (op_st_dc_dn & 0xFC);
       op_st_dc_dn = (op_st_dc_dn | 0x02);
       }

     if(command == 0xAB)
       {
       op_st_dc_dn = (op_st_dc_dn & 0xFC);
       op_st_dc_dn = (op_st_dc_dn | 0x03);
       }
```

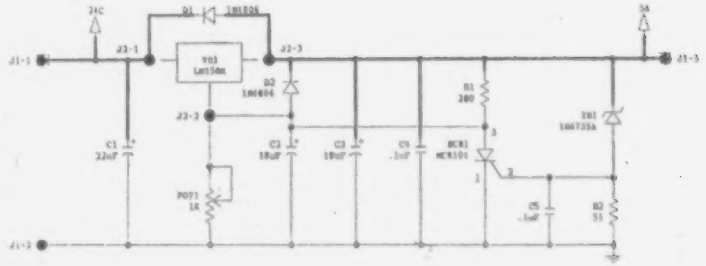
```
        outp(BASE_ADDR + 0x02, op_st_dc_dn);        //set output lines to the
        test_alarms();                              //desired modulation
        return;
}


void set_doppler_frequency(void)

{  // function to set the square wave clocks to the 74ALS163 counters and
   // the MAX293 low pass filter.  The counter clock is
512*doppler_frequency_int
   // and the filter clock is 130*doppler_frequency_int. Counter 0 is used for
   // the filter, counter 1 is used for the counters for the EPROM address.
   // address of counter control register is 271 (0x100+12+3)
   // address of counter 0 data is 268 (0x100+12+0)
   // address of counter 1 data is 269 (0x100+12+1)
   // the 5300 board must have been modified to make counter 1 independent
   // and bring its output out to J1-21 in place of CTC302.
   // doppler_frequency_int is a global unsigned int variable representing
   // the doppler frequency from 10 Hz to 7812 Hz.

  counter_clock = 0;
  filter_clock = 0;
  counter_0_value = 0;
  counter_0_value_int = 0;
  counter_1_value = 0;
  counter_1_value_int = 0;

  counter_clock = 512 * doppler_frequency_int; // calculate frequency for DAC
  filter_clock =  120 * doppler_frequency_int; // low pass filter frequency is
                                    //1.3 times the doppler frequency
  counter_0_value = _8MHz / filter_clock; // get divisor for counter 0 and
  counter_0_value_int = counter_0_value;  // round it off to the nearest
integer
  if ((counter_0_value-counter_0_value_int) > 0.499) counter_0_value_int++;
  counter_1_value = _8MHz / counter_clock; // get divisor for counter 1 and
  counter_1_value_int = counter_1_value;  // round it off to the nearest
integer
  if ((counter_1_value-counter_1_value_int) > 0.499) counter_1_value_int++;
  outportb(271,54);     // write control word for counter 0= 00110110 (mode 3)
  outportb(268,(counter_0_value_int & 0xFF));      // write counter 0 LSB
  counter_0_value_int = counter_0_value_int >> 8; // shift right 8 bits
  outportb(268,counter_0_value_int);               // write counter 0 MSB
  outportb(271,118);    // write control word for counter 1= 01110110 (mode 3)
  outportb(269,(counter_1_value_int & 0xFF));      // write counter 1 LSB
  counter_1_value_int = counter_1_value_int >> 8; // shift right 8 bits
  outportb(269,counter_1_value_int);              // write counter 1 MSB
  test_alarms();
  return;
}
```

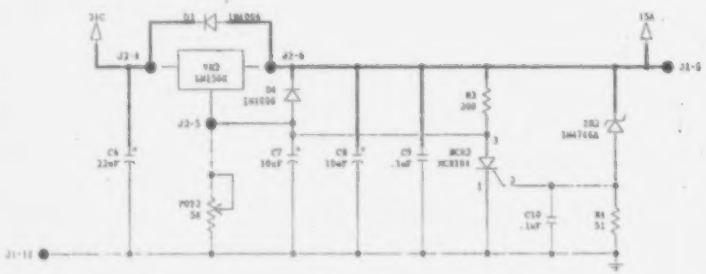
```
void send_status(void)
      {
         byte_1 = RF_frequency;
         sendcom(0, byte_1, 0);

         byte_2 = atten_RF_en;
         byte_2 = byte_2 >> 1;
         if(atten_RF_en & 0x01)
              {byte_2 = byte_2 | 0x80;}
         if(loc_rem_bit & 0x10)
              {byte_2 = byte_2 | 0x40;}
         sendcom(0, byte_2, 0);

         if(op_st_dc_dn & 0x01)
              {byte_3 = byte_3 | 0x40;}
         if(op_st_dc_dn & 0x02)
              {byte_3 = byte_3 | 0x80;}
         byte_3 = byte_3 | least_5;
         sendcom(0, byte_3, 0);

         byte_4 = most_8;
         sendcom(0, byte_4, 0);


       test_alarms();
       return;
      }
```

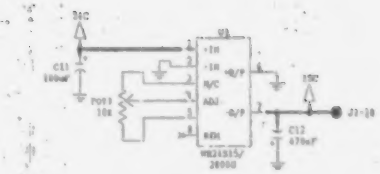
APPENDIX B

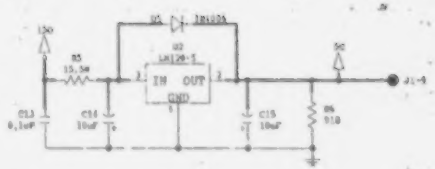
CIRCUIT SCHEMATICS

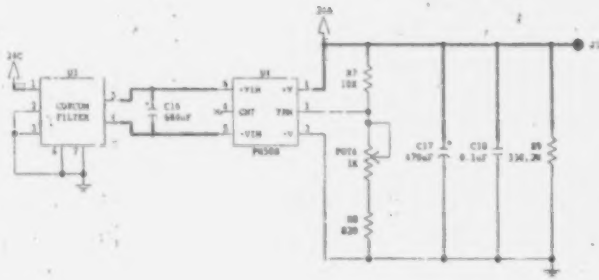
[illegible] (c) Department of Industry 1995
[illegible]
[illegible] (c) Minister of Supply and Services Canada 1995
[illegible]



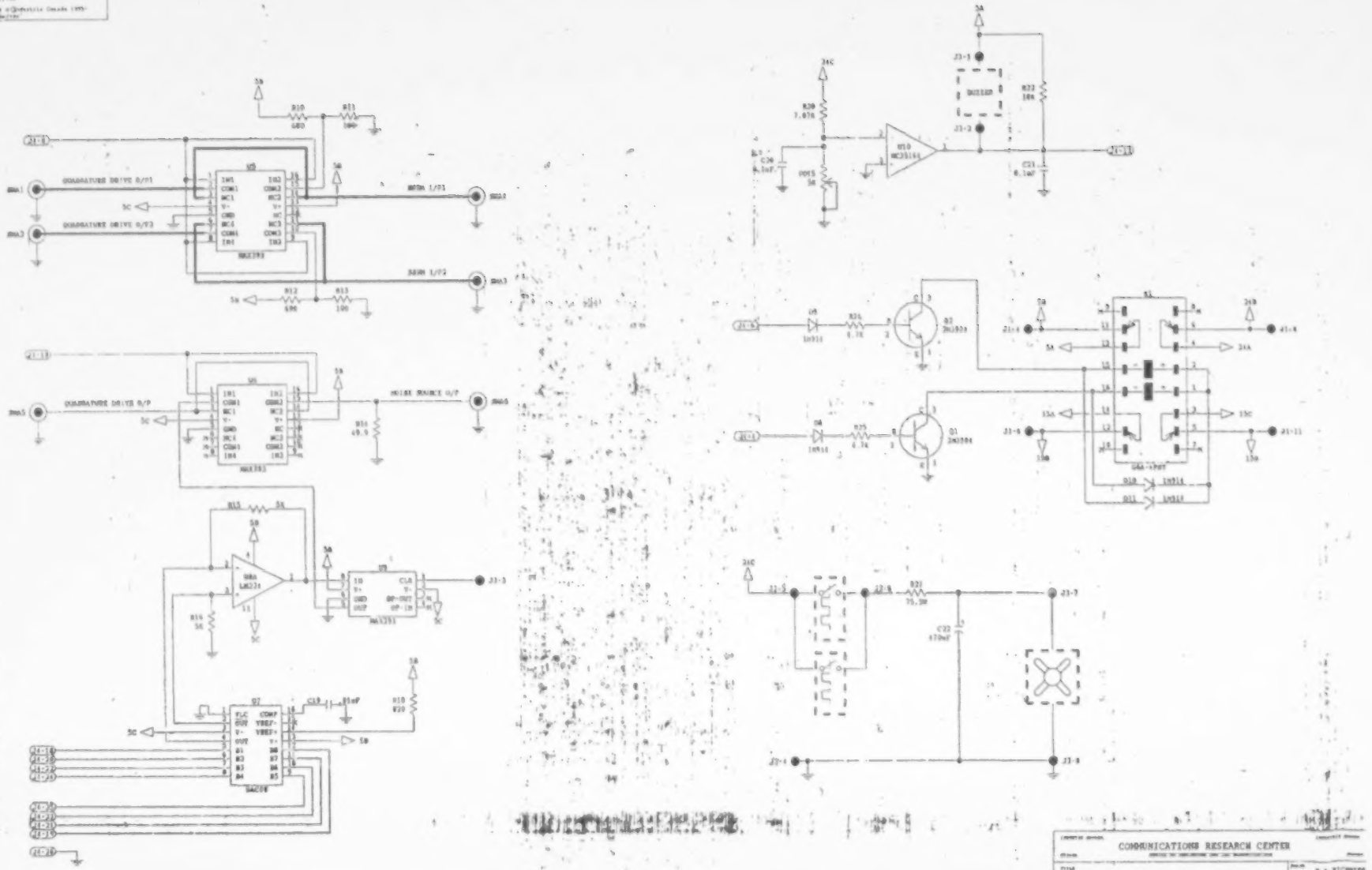


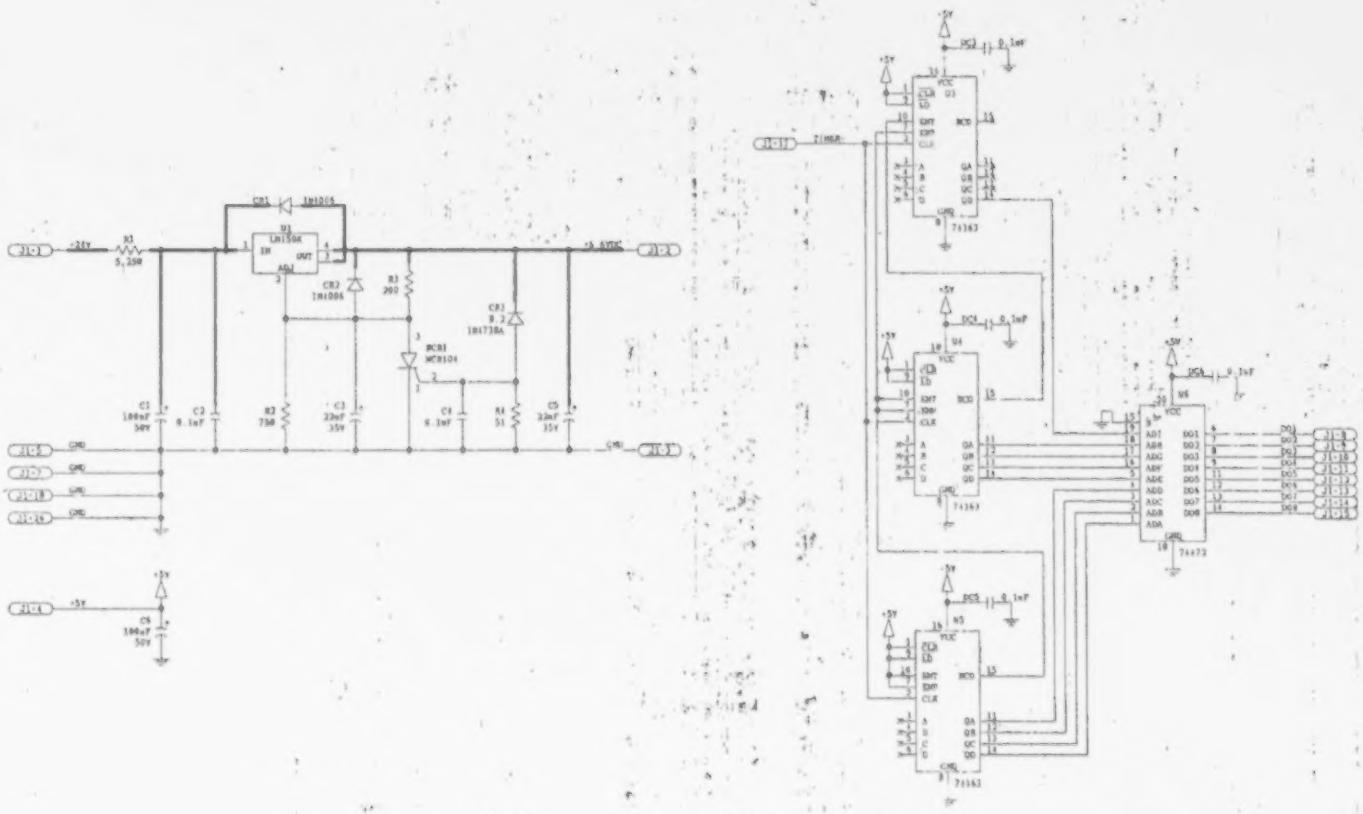






			COMMUNICATIONS RESEARCH CENTER			
			PIPS POWER SUPPLY BOARD			
NO.	DATE	REVISION	DRAWING NO.	CAD FILE NO.	DRAWN BY	
			95-042	DRED-BARAR	3-16536-0	


COMMUNICATIONS RESEARCH CENTER
PIPS POWER SUPPLY BOARD
JUNE 1995
DRAWING NO. 8-14530-0



COMMUNICATIONS RESEARCH CENTER

PIPS DAUGHTER SCHEMATIC

R.A. RICHARDS

JULY 1995

95-077

RADAR-PWD

8-16536-0

SECURITY CLASSIFICATION OF FORM
(highest classification of Title, Abstract, Keywords)

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.)

APPLIED SILICON INC., CANADA
220-2427 HOLLY LANE
OTTAWA, ONTARIO KIV 7P2

2. SECURITY CLASSIFICATION (overall security classification of the document including special warning terms if applicable)

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C or U) in parentheses after the title.)

PORTABLE INTERFERENCE FREQUENCY SOURCES
REMOTE CONTROL UPGRADE FINAL REPORT

4. AUTHORS (Last name, first name, middle initial)

CURNOW, JOHN W.

5. DATE OF PUBLICATION (month and year of publication of document)

AUGUST 1995

6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc.)

40

6b. NO. OF REFS (total cited in document)

2

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

FINAL TECHNICAL REPORT

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.)

DND - DEFENCE RESEARCH ESTABLISHMENT OTTAWA
RADAR AND SPACE DIVISION (SURFACE RADAR)
SHIRLEY'S BAY, OTTAWA, ONTARIO

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant)

EARS RADAR RESEARCH PROJECT

9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written)

W7714-5-9868

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)

974-55126-02

10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor)

DRBO CONTRACTOR REPORT 98-623

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(X) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availabilty (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

UNLIMITED

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both offical languages unless the text is bilingual).

This report presents the final test results for the Portable Interference Frequency Sources (PIFS) following the modifications to enable remote control of their functions. The PIFS are two identical units which serve as radar beacons for the Experimental Array Radar System (EARS). They operate over the same 8.900 to 9.400 GHz bandwidth that the EARS transmits on. They can transmit a continuous wave (CW) signal. They can also modulate the CW signal with a low frequency doppler or a 10 MHz bandwidth noise source. The system controller (syscon) can control the transmit frequency, the attenuation, the output on/off enable, the modulation type and the doppler frequency of the PIFS. Each PIFS can be controlled independently, with two in operation simultaneously. In addition, the syscon can request the PIFS status and put the PIFS into a standby mode, to conserve battery power. A small hand-held VT-100 compatible terminal is used at each PIFS for local control and display of the status. Power is derived from two gell-cell batteries, contained in a separate enclosure. The PIFS can operate at up to 15 kilometers away from the EARS, depending on the antennas used for the radio link and their orientation to each other.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

EARS
beacon source
system controller
remote control
control-program
PIFS
jammer
radio modem
hand-held terminal